Génération de code

Université Libre de Bruxelles
2008 - 2009

Denis BOIGELOT Sébastien COLLETTE Gilles GEERAERTS

Génération de code

- Quel code générer ? Pour quelle architecture cible ? **P-langage pour une P-Machine**
- À quel moment de l'analyse faut-il le générer ? Comment spécifier formellement cela ? **Action sémantique** et **Grammaires attribuées**

P-langage

- Dans le cadre des TPs et du projet, nous allons générer du P-langage.
- Langage intermédiaire développé pour la compilation de Pascal, interprété par une P-Machine
- C'est une machine à stack

Machine à stack ?

- Idée : Pas de registres de travail, on utilise un stack à la place.
- Exemple réaliser la somme de 4 et 5
- Dans une machine avec registres :

  mov AX, 4 mov BX, 5 add AX, BX

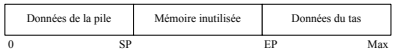
  le résultat est dans AX.
- Dans une machine à pile :

  push 4 push 5 add

  le résultat est au sommet du stack.

Mémoire et registres

- La mémoire est divisée en deux parties.
- **Zone de code** contient le programme. Le registre PC indique l'adresse de la prochaine instruction.
- **Zone de données** divisée en deux parties :
  - le stack va de l'adresse 0 (bas) à l'adresse stockée dans le registre SP (compris);
  - le heap va de l'adresse contenue dans le registre EP à la fin de la zone.



P-Machine

- P-Machine développée aux FUNDP (http://www.info.fundp.ac.be/~gpm/)
- Licence GPL, interface pratique, cross-platform (Java)
- Mode d'emploi sur la page des TPs
- Cellules du stack et du heap typées (entier ou booléen)
- Code à exécuter fourni dans un fichier texte
- lignes de commentaires = lignes commençant par ;

Instructions du stack

- **ldc i q** - Push de la constante entière q
- **ldc b q** - idem avec booléen
- **ldo i a** - Push de l'entier qui est à l'adresse a
- **ldo b a** - idem avec booléen
- **sro i a** - Pop et met la valeur à l'adresse a
- **sro b a** - idem avec booléen
- **pop** - Pop

Instructions arith. et logiques

- **div i** - Pop a et b, et push b/a
- **mul i** - idem avec b $*$ a
- **add i** - idem avec b + a
- **sub i** - idem avec b – a
- **and b** - idem avec a $\wedge$ b
- **or b** - idem avec a $\vee$ b
- **neg i** - Pop a et push –a
- **not b** - Pop a et push ¬a

a et b doivent être entiers pour div, mul, add, sub, neg, et booléens pour and, or, not

Instructions d'I/O

- **prin** - Affiche la valeur entière au sommet de la pile sur la sortie et la supprime de la pile.
- **read** - Lit une valeur entière sur l'entrée et la place au sommet de la pile.

Exercice 1

Écrivez le code qui calcule et affiche la valeur de :

$$(3 + x) * (9 - y)$$

où x est une valeur lue sur input et y est la valeur qui se trouve en mémoire à l'adresse 0 (on verra plus tard comment placer correctement une valeur en mémoire).

Solution 1

```
; calcul de 3 + x
ldc i 3
read
add i
; calcul de 9 - y
ldc i 9
ldo i 0
sub i
; calcul du produit
mul i
; affichage
prin
```

Suppose qu'une valeur a été correctement placée à l'adresse 0.

Comparaisons

- **geq i** - Pop a et b et push la valeur booléenne correspondant à $b \geq a$
- **leq i** - Même chose avec $b \leq a$
- **les i** - Même chose avec $b < a$
- **grt i** - Même chose avec $b > a$
- **equ i** - Même chose avec $b = a$
- **neq i** - Même chose avec $b \neq a$

a et b doivent être entiers

Branchements

- **define L** - Place l'étiquette L
- **ujp L** - Saut non conditionnel. La prochaine instruction sera celle qui suit define L
- **fjp L** - Saut conditionnel. Pop un booléen. Si faux, la prochaine instruction sera celle qui suit define L. Sinon, continue normalement

Exercice 2

Écrire le code qui affiche toutes les valeurs impaires dans l'intervalle [7; 31]. Pour ce faire vous aurez besoin de l'instruction **dpl i** qui duplique le sommet (entier) de la pile.

Solution 2

```
ldc i 7
;debut boucle
define debutboucle
; test boucle
dpl i
ldc i 31
leq i
fjp fin
; corps boucle
dpl i
prin
ldc i 2
add i
ujp debutboucle
; fin boucle
define fin
stp
```

Exercice 3

Donner la structure d'un if-else en P-langage.

Solution 3

…	évaluation de la condition
fjp else	saut si condition fausse
…	code then
ujp fin	pour éviter le else
define else	
…	code else
define fin	fin du if

Gestion mémoire

- Quand on a des variables statiques dans le programme source, il faut se réserver de la place sur le stack pour stocker leurs valeurs.
- L'instruction **ssp q** donne au stack la taille q.
- À n'appeler qu'une fois au début du programme sinon on perd le contenu du stack.
- Quand on appelle **ssp q** au début du programme, on se réserve donc les cases d'adresses 0, …, q − 1.

Exercice 4

Écrire le code qui :

- se réserve de la place pour deux variables que nous appellerons a et b ;
- initialise a et puis b avec des valeurs lues sur input ;
- ajoute 5 à a ;
- divise b par 2 ;
- si a > b, affiche a, sinon affiche b.

Tout au long de l'exercice, les valeurs stockées dans les cases correspondant à a et b devront rester cohérentes.

Solution 4

```
; a == adresse 0
; b == adresse 1
ssp 2
; Initialisation
read
sro i 0
read
sro i 1
```

```
; a+=5
ldo i 0
ldc i 5
add i
sro i 0
; b/=2
ldo i 1
ldc i 2
div i
sro i 1
```

```
; a > b
ldo i 0
ldo i 1
grt i
fjp else
ldo i 0
prin
ujp fin
define else:
ldo i 1
prin
define fin:
stp
```

Analyse sémantique

- Il reste à faire le lien avec les grammaires…
- **Actions sémantiques** actions rajoutées dans la grammaire. Effectue les actions lors de l'analyse sémantique.
- **Grammaire attribuées** consistent à évaluer des attributs associés aux noeuds de l'arbre syntaxique.

Actions sémantiques Exemple

- La grammaire suivante accepte les expressions en NPI:

$$S \rightarrow id$$
$$\rightarrow SS*$$
$$\rightarrow SS+$$

- On peut traduire ces expressions en notation algébrique directe (NAD) à l'aide des actions sémantiques:

$$S \rightarrow id \ \{printf(val(id);\}$$
$$\rightarrow S \ \{printf("*");\} \ S*$$
$$\rightarrow S \ \{printf("+");\} \ S+$$

Exercice 5

- Voici une règle pour définir le if d'un langage impératif :

  ⟨Si⟩ → if ⟨Cond⟩ then ⟨Code⟩ ⟨SuiteSi⟩

  ⟨SuiteSi⟩ → else ⟨Code⟩ endif

  ⟨SuiteSi⟩ → endif

- Indiquez quelles instructions en P-langage vous devez générer pour traduire ce code dans un compilateur en descente récursive. Faites l'hypothèse que les règles correspondant à ⟨Cond⟩ et ⟨Code⟩ sont déjà décorées pour générer le code correct.

Solution 5

cpt est un compteur global initialisé à 0.

⟨Si⟩ → if ⟨Cond⟩	{*cpt*++}
	{Produire(fjp else*cpt*)}
then ⟨Code⟩⟨SuiteSi⟩	
⟨SuiteSi⟩ → endif	{Produire(define else*cpt*)}
→ else	{Produire(ujp fin*cpt*)}
	{Produire(define else*cpt*)}
⟨Code⟩ endif	{Produire(define fin*cpt*)}

Grammaires attribuées

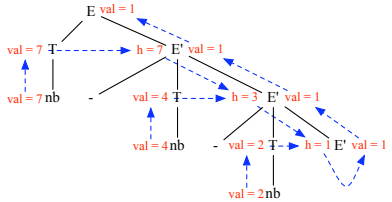
- On distingue deux types d'attributs :
- **Attributs hérités** la valeur dépend des valeurs des attributs du père et des frères d'un nœud.
- **Attributs synthétisés** la valeur dépend des valeurs des attributs des fils d'un nœud ou, si le noeud est une feuille, la valeur est donné par l'analyseur lexical.
- Les grammaires S-attribuées n'ont que des attributs synthétisés, par exemple: *yacc*.

Exemple

- Considérons un fragment de grammaire qui permet de calculer des expressions arithmétiques.

$E \rightarrow TE'$	$\{E'.h \leftarrow T.val\}$
	$\{E.val \leftarrow E'.val\}$
$E' \rightarrow -TE'_1$	$\{E'_1.h \leftarrow E'.h - T.val\}$
	$\{E'.val \leftarrow E'_1.val\}$
$E' \rightarrow \epsilon$	$\{E'.val \leftarrow E'.h\}$
$T \rightarrow nb$	$\{T.val \leftarrow nb.val\}$

- 7 - 4 - 2 sera évalué:



Exercice 5

1. Réécrivez la grammaire suivante en tenant compte de la priorité et de l'associativité des opérateurs et associez à cette grammaire les règles et attributs nécessaires pour calculer la valeur d'une expression E.

$$E \rightarrow E \text{ op } E \mid (E) \mid nb$$

$$op \rightarrow + \mid - \mid * \mid /$$

2. Dérécursifiez cette grammaire et adaptez les règles.

Solution 5.1

$E \rightarrow E_1 + T \quad \{E.val \leftarrow E_1.val + T.val\}$

$\rightarrow E_1 - T \quad \{E.val \leftarrow E_1.val - T.val\}$

$\rightarrow T \quad \{E.val \leftarrow T.val\}$

$T \rightarrow T_1 * F \quad \{T.val \leftarrow T_1.val * F.val\}$

$\rightarrow T_1 / F \quad \{T.val \leftarrow T_1.val / F.val\}$

$\rightarrow F \quad \{T.val \leftarrow F.val\}$

$F \rightarrow (E) \quad \{F.val \leftarrow E.val\}$

$\rightarrow nb \quad \{F.val \leftarrow nb.val\}$

Solution 5.2

$E \rightarrow T \ \{E'.h \leftarrow T.val\}$
$\quad E' \ \{E.val \leftarrow E'.s\}$
$E' \rightarrow +$
$\quad T \ \{E'_1.h \leftarrow E'.h + T.val\}$
$\quad E'_1 \ \{E'.s \leftarrow E'_1.s\}$
$\rightarrow -$
$\quad T \ \{E'_1.h \leftarrow E'.h - T.val\}$
$\quad E'_1 \ \{E'.s \leftarrow E'_1.s\}$
$\rightarrow \varepsilon \ \{E'.s \leftarrow E'.h\}$

Solution 5.2

$T \rightarrow F \ \{T'.h \leftarrow F.val\}$
$\quad T' \ \{T.val \leftarrow T'.s\}$
$T' \rightarrow *$
$\quad F \ \{T'_1.h \leftarrow T'.h * F.val\}$
$\quad T'_1 \ \{T'.s \leftarrow T'_1.s\}$
$\rightarrow /$
$\quad F \ \{T'_1.h \leftarrow T'.h / F.val\}$
$\quad T'_1 \ \{T'.s \leftarrow T'_1.s\}$
$\rightarrow \varepsilon \ \{T'.s \leftarrow T'.h\}$
$F \rightarrow (E) \ \{F.val \leftarrow E.val\}$
$\rightarrow nb \ \{F.val \leftarrow nb.val\}$